Enabling Continuously Evolving Context Information in Mobile Environments by Utilizing Ubiquitous Sensors

Stefan Forsström and Theo Kanter

Mid Sweden University, Sundsvall 851 70, Sweden
{stefan.forsstrom,theo.kanter}@miun.se

Abstract. Context-aware applications require local access to current and relevant views of context information derived from global sensors. Existing approaches provide only limited support, because they either rely on a network broker service precluding open-ended searches, or they adopt a presence model which has scalability issues. To this end, we propose a fully distributed architecture employing context user-agents co-located with data-mining agents. These agents create and maintain local schemas using ranking of global context information based on context proximity. Continually evolving context information thus provides applications with current and relevant context views derived from global sensors. Furthermore, we present an evaluation model for assessing the effort required to present local applications with current and relevant contextual views. We show in a comparison with earlier work that the approach achieves the provisioning of evolving context information to applications within predictable time bounds, circumventing earlier limitations.

Keywords: Evolving context, ubiquitous sensors, mobile environments.

1 Introduction

With the current escalation within mobile Internet-access and smart mobile devices, users demand applications to behave more intelligently. One group of such intelligent applications is called context-aware applications. These applications are made aware of their user's context, to change their own behavior. This opens up a new field of user friendly services, which can be per person adapted to provide the best possible service for that particular user. These applications do however require reliable context information to perform intelligent decision making. This context information have traditionally been gathered from stored personal preferences or local sensors on each end users device. However, some applications have already identified the advantages of also utilizing context information from other users, to create even better collaborative services. Because of this, we focus on scenarios which demand a massive collaborative user base and applications which require access to a large amount of continuously changing context information. These scenarios are for example, the exchange of road

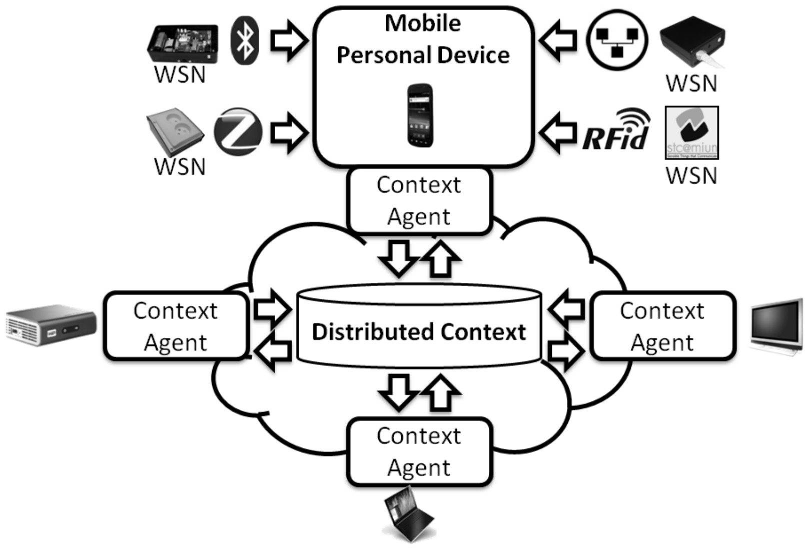

Fig. 1. Our approach to the Internet of Things

characteristics and environmental information between a large number of cars, when traveling at high speed during hazardous weather. Or the enabling of context based advertisements when users are near a shopping mall. For example, receiving notification on the amount of available parking space, relevant sales based on the wardrobe of the user, and the availability of the users clothing size in the store. Furthermore, these scenarios also include social applications, for example finding other users with similar interests, or notification on status changes of relevant friends based on their situation.

To address this, we envision a system where a wide range of connected devices acts as routers and gateways, for sensor based context information coming from wireless sensor networks (WSN). This can then be shared on the Internet, creating a system for distributed context, called the Internet of Things. Figure 1 shows an overview of this general system which contains devices connected to the Internet, sharing and relaying context information from a wide range of attached sensor in a distributed support. Furthermore, in this paper we will examine the possibility to create continuously changing context information that can be utilized in this general architecture. This is motivated by the fact that we will require intelligent context-aware services in the future Internet of Things, which will demand usage of context in a manner similar to how context behaves in the real world. Therefore, we consider context information to be continuously changing and evolving over time. To be concrete, the location of a mobile entity can change multiple times per second and because other entities services might utilize this information, the most current location must be made available without delay. The same logic applies to other sensor based context, such as temperature, humidity, proximity, but also other context information such as mood, interests, personal profile, etc. Furthermore, intelligent applications will require the context to be current, reliable, and provided without delay even when performing an open ended search. This complexity poses some significant problems, since context changes very quickly and sometimes without prior notion. We will therefore need a system which can manage this constantly changing and evolving context, in addition to disseminating and exchanging the context information to other entities within predictable time bounds. In detail, each entity in such a system is required to have a constantly changing object, representing their current context which must be continuously updated and made available to applications, in the form of a view of the entity's current context. Therefore, this paper focuses on the problem of creating a system which is capable of managing this context evolution, in a manner which can be utilized in mobile applications without hindering the information flow. To be concrete, the following requirements must be supported:

1. Ubiquitous access to global context information derived from a large number of sensors with continuously changing values.
2. Support a large number of mobile entities and avoiding single points of failures.
3. Dissemination of context information within predictable time bounds.
4. Open ended searches in the system, without prior knowledge of the context sources.
5. Provide access to continually changing global context information as current, relevant, and accurate contextual views.

The remainder of this paper is organized in the following way: Section 2 examines background and related work in the area. Section 3 presents our approach and proposed system for evolving context information. Section 4 presents our proof of concept prototype and evaluation. And section 5 presents our conclusions and future work.

2 Related Work

Context and context awareness have been studied for some time in relation to context-aware applications. But with the recent escalation in the mobile market, the amount of context-aware applications for mobile devices have proliferated and increased in multitude. The term context have been defined and redefined on multiple occasions. One of the most well known within computer science defines context as the elements of the user's environment which the computers knows about. But this was later redefined to also include circumstances, situation, and relevance. Hong et al. [1] has studied this in detail and the problems which are faced when creating context awareness. Furthermore, the constant change of context information dictates that it must be gathered from many different sources concurrently at the same time. This of often achieved by using automated sensors, predefined personal profiles, schedules, social networks, address information, statistics, etc. This autonomously acquired context information can then be used to create context awareness, which has been formalized into what we today see as context-aware applications.

2.1 Related Context Exchange Systems

Related systems can be organized into three distinct categories, depending on where they store the actual context information. In detail, these three categories are: centralized, semi distributed, and fully distributed storage.

Centralized systems store the context information under a single administrative authority, either in a single large database or replicated in a cloud based manner. The IMS presence system [2] is an example of such a centralized system, which provides presence services that are governed and administrated by an operator. Other examples of centralized storage include the SenseWeb project [3], SENSEI project [4], and the presence extension to XMPP [5]. Project SERENOA [6] also has a similar centralized approach with ontology based modeling of context. However, all of these centralized systems have scalability issues when the number of updates and queries in the centralized storage increase in magnitude. This will happen when the number of connected device increases to the magnitude of millions of entities. Therefore, they will have problems to support requirement 1 concerning continuously updating data from a large number of entities. Furthermore, centralized systems also have problems with requirement 2, because they expose a single points of failure.

Semi distributed systems store the context information locally on each entity in a peer-to-peer manner, but still maintaining a centralized authority for the exchange of context between peers. These systems use session establishment protocols such as SIP to exchange the context, but under supervision by a centralized authority. The Mobilife project [7], the CONTEXT project [8], and the ADAMANTIUM project [9] are examples of such a system with peer-to-peer exchange of context under centralized supervision. Naturally, these systems scale better in comparison to centralized systems. But they still maintain a centralized component, which will become a bottleneck for the context exchange when the entities perform open ended queries on a large and continually changing dataset. This comes from the fact that the centralized component has to administrate all the sessions, even if the actual exchange is performed outside of the centralized component. Therefore, semi distributed systems have problems addressing the single point of failure of requirement 2, because of the centralized component. In addition to having problems with the support for the open ended searches in requirement 4.

Fully distributed systems both stores and administrates the context locally on each entity in a fully peer-to-peer oriented manner. These systems often utilize distributed hash tables to enable logarithmic scaling when the number of entities increases in magnitude. Examples of such systems are the MediaSense framework [10], the SOFIA project [11], and COSMOS [12]. Naturally, these systems does not contain any single point of failure and are thus more resilient, even if the distribution itself often require additional overhead for maintaining the overlay. Also, they have no real support for evolving context and the provision of contextual views to applications, which is demanded by requirement 5.

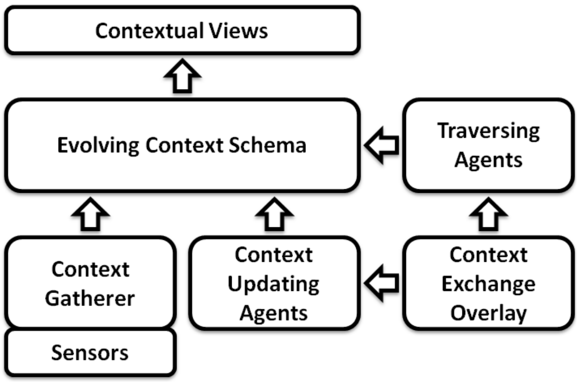

Fig. 2. Overview of the proposed architecture

3 Creating Evolving Context Schemas

Because the related systems do not support all the previously specified requirements, a new approach is required. Therefore, this paper present an approach which builds on previous results from [13] and extends that initial approach to enabling ad hoc context exchange in mobile devices. But in this paper we consider context to be following a natural form of progression and evolution, which will gradually change with the situation of the user. We will represent this evolving context information by an information object called an evolving context schema, which contains all context information related to a specific entity. This follows the Context Information Integration (CII) model explained in [14], which proposes the use of a context schema that combines oriented models with ontological properties. This context schema changes with the user, and contains all relevant contextual information about that particular entity. Furthermore, in this paper we will adapt this context model to support dynamic and evolving context views, created from context schemas located on each entity's end device. In detail, the schema will be transformed from a stable database element as it was in the CII model, to a continuously changing object with the most current information that can provide context views to applications within predictable time bounds. An overview of our proposed architecture can be seen in figure 2. Each of these parts will be explained further in detail on how they contribute to the creation and utilization of evolving context information. But in overview, our architecture contains the evolving context schema, the context gatherer, the traversing agents, the context updating agents, and the context exchange overlay network.

3.1 Evolving Context Schema

Context schemas will be used to model the evolving context information in our system. But to create these evolving context schemas, we will require relations between different entities' context. In order to solve this, we utilize context proximity calculations to find related entities based on the context proximity distance of their respective context schemas. This means that it is possible to determine that two entities are connected and thus related to each other, based on the fact that their context relates to each other by a predefined context proximity distance. This proximity in regards to context has been described by Antifakos et al. [15], although at that point it was quite simple and primitive. But in detail, it builds on the idea that if two entities who share similar context is probably relevant to each other, and is therefore within context proximity to each other. Furthermore, it is very difficult to determine the context proximity between non-scalar context values, such as favorite color or favorite restaurant. And because of this, we will assume in this paper that context proximity can be calculated by a universal context proximity function. However, we acknowledge that it will be problematic calculating this scalar distance in the general case between all different types of context. This schema evolution with relations based on context proximity can be illustrated algebraically as in equation 1 and 2. In these equations, $C_{evolving}$ is the set of currently evolving context in the context schema for a particular entity. This evolving context schema is defined as the union of the set of all local context C_{local} and the set of all relevant remote context $C_{relevant}$. Furthermore, this relevant set is determined by $f_{cp}()$, a context proximity function that can determine if the remote context C_{remote} is relevant, based on a specified proximity distance d.

$$\{C_{evolving}\} = \{C_{local}\} \cup \{C_{relevant}\} \tag{1}$$

$$\{C_{relevant}\} = \{C_{remote} : f_{cp}(C_{remote}) \leq d\} \tag{2}$$

These context schemas will continually be evolved by the system, as the entities' context is always changing. Thus in our architecture, the context gatherer and the updating agents evolves context from local and remote sources, thus keeping the C_{local} and the $C_{relevant}$ set up to date. While the traversing agents traverses the network to find new remote context information, that can be considered relevant based on the context proximity function. Lastly, the context schema provides contextual views to applications from the evolving schema. In relation to the scenario with cars exchanging road characteristics, this will mean that a car's context schema would contain the context of the other cars traveling on the same road as well as the context of its driver and all passengers. Thus exchanging context with other relevant cars, to provide contextual views to applications running in their driver assist systems. Hence, the evolving context schema addresses requirement 5 regarding context views, because applications can access the evolving context schema and get a view of the current context. Furthermore, the evolving context schema also addresses requirement 4 regarding open ended searches, because of the context relations created by the context proximity function.

3.2 Context Gatherer

The context gatherer is a component which gathers context from local sources, thus providing the C_{local} set in equation 1. This component is required for the enabling of evolving context since most context information has a local sensor as its originating source of contextual data. These sensors do however impose their own problems, since they only provide raw values and often in many different formats based on each manufacturer. Thus, the problem that the context gatherer solves is to create context information from many different types of sensors, while providing this context information upward to the evolving context schema. In detail, the context gatherer continually read values from sensors either locally attached, built-in, or connected trough local wired and wireless networks, to subsequently update the values into the evolving context schema. Following this, we propose the usage of multiple concurrent gatherers which should be specialized for the different types of sensors and their specific update frequency. Hence, the context gatherer contributes to the addressing of requirement 1 regarding global access to sensor information, because it provides sensor information as context into the evolving context schemas. The operation of the context gatherer can be seen as a flow chart in figure 3. Furthermore, pseudo code for the context gatherer's operation can be seen below.

```
loop
  wait for sensor update
  if new sensor value then
    update context schema
  end if
end loop
```

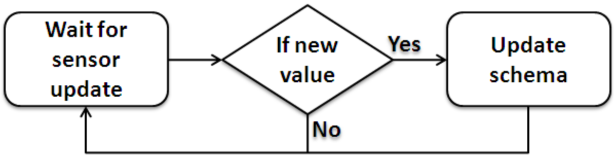

Fig. 3. Operation of the context gatherer

3.3 Traversing Agents

The traversing agents solve the problem of finding new relevant context, thus providing the $C_{relevant}$ set in equation 2. In detail, the traversing agent browses the local context schema for relations which could be relevant to explore. To find a new entity, the traversing agent communicates with a known entity from the local context schema, asking for other relevant entities. To determine if another entity is relevant, the traversing agent utilizes the context proximity function with a predefined distance. If two entities are in context proximity to each other, they are considered relevant and the other entity's relevant context is inserted into the local context schema. Because of all the traversing, these agents operate on a best effort system, always traversing the network and evaluating other entity's context. Each entity can have multiple traversing agents operating at the same time, because they can concurrently expand the evolving context schema without internal interference. The traversing agent contributes to the addressing of requirement 1 regarding global access to sensor information, because it enables exchange of ubiquitous context information. The algorithm for a traversing agent is a five step process. Firstly, it acquires the local context schema. Secondly, it examines the local context schema and chooses a relation which it wants to traverse. Thirdly, it traverses this relation, communicating and fetching the remote entity's schema. After it has both the local and the remote schema, it performs an evaluation based on the context proximity between the schemas. Depending on the result of the context proximity evaluation, it determines which parts of the remote context information should be included in the local context schema. This algorithm for the traversing agents can be seen in figure 4 and pseudo code for the traversing agent's operation can be seen below.

```
loop
  get local context schema
  choose a relation
  acquire remote context schema
  for all context in remote schema do
    if is within context proximity then
      add to local context schema
    end if
  end for
end loop
```

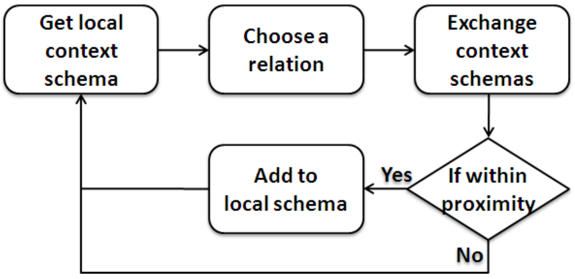

Fig. 4. Traversing agent algorithm

3.4 Context Updating Agents

The context updating agents operate in a similar manner as the context gatherer. The main difference is that the content gatherer acquire context from local sources, and the context updating agents acquire context from remote sources. Therefore the context updating agents is responsible for keeping the $C_{relevant}$ set in equation 1 continuously updated and accurate. The context updating agents are required because the traversing agent only finds new relations, they do not keep the context values continuously updated. In detail, the context updating agents examines the local schema and determines if a context value requires updating. If this is the case, it establishes a connection to the remote source and acquires the most recent value, to then update the local schema. The system will require multiple context updating agents for keeping all context values continuously updated. Therefore, multiple context updating agents will run concurrently and acquire context from many different sources at the same time. Hence, the context updating agent contributes to the addressing of requirement 1 regarding global access to sensor information, because it maintains the context from sensors with continuously changing values. In detail, the algorithm for a context updating agent can be seen in figure 5 and visualized in pseudo code below.

```
loop
  get local context schema
  choose a context value
  if value require update then
    acquire remote context schema
    if new value is within context proximity then
      update local context with new value
    else
      remove context value from local context schema
    end if
  end if
end loop
```

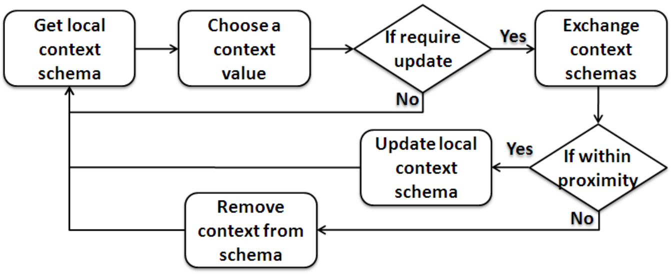

Fig. 5. Context updating agent algorithm

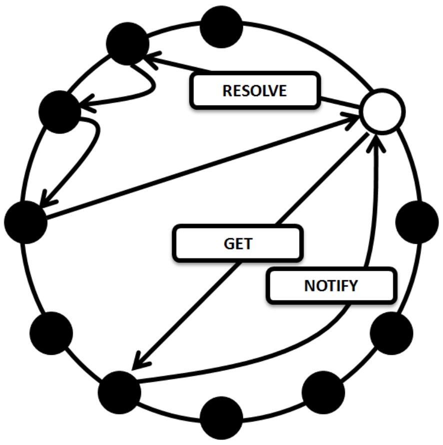

Fig. 6. DCXP protocol operation

3.5 Context Exchange Overlay

The communication between entities must be done in a scalable and dynamic manner, without inducing unnecessary delay. These requirements demand the usage of a peer-to-peer oriented protocol, which enables direct connections between entities. This can be realized by any infrastructure able to provide scalable context dissemination within predictable time bounds, but we have realized it with the usage of a completely distributed context exchange system called the DCXP network [16]. However, any type of scalable context exchange network can be used to create evolving context, even cloud based storage, presence services, or session establishment systems. In detail, DCXP enables direct dissemination of context between entities that have joined an overlay network, using a context user-agent. An overview figure of the dissemination of the DCXP network can be seen in figure 6, which shows how a context user-agent first must resolve a sought after context identity and then get the value from the remote source. The DCXP network scales well due to the logarithmic lookup in its distributed hash table. But the hash table can be exchanged to other similar infrastructure if demanded, for example P-grid. Also, because the actual dissemination is performed on a peer-to-peer basis without proxies, network delay is kept to a minimum. This peer-to-peer communication is paramount to the continual context evolution, because both the context updating agents and the traversing agents operate under the premise that communication is performed with minimized delays and within predictable time bounds. Furthermore, the DCXP network can run without relying on centralized naming services such as DNS. The DCXP network also provide the option to perform open ended searches utilizing the relations between entities on the overlay, which was demanded by requirement 4. The DCXP network also address requirement 2 about large amounts of entities without central point of failures and requirement 3 about predictable time bounds.

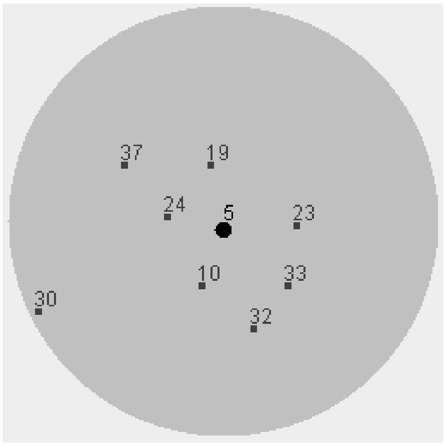

Fig. 7. A view from the evolving context schema for entity number five

This is because the resolving in the DCXP system scales logarithmically and that the context is exchanged on a peer-to-peer basis after the initial resolve.

4 Prototype and Evaluation

Figure 7 presents a running proof of concept prototype that shows our approach in a simulated environment. In detail, the figure shows the graphical view of an entity's evolving schema. The graphical view is centered on a particular entity, in this case entity number five. The view contains the other entities that are in context proximity, as well as the context proximity distance which can be seen as a circle in the background. This view is updated continuously as the entity move around and as new entities enters or leaves the context proximity area. This simulation is comparable to the scenario with the traveling cars exchanging road characteristics, as the entities constantly move around and thus dynamically exchange context information between each other. The schema in the prototype does however only contain location based context values, in the form of latitude and longitude. But in conclusion, the prototype proves the feasibility of the approach because it creates evolving context information from sensor based context. Furthermore, the proposed system was evaluated in comparison to related alternative systems. And because the related systems were categorized into three different types, centralized, semi distributed, and fully distributed systems, the proposed system will be compared against those general architectures.

4.1 Comparison to Centralized Architectures

The scaling of resources such as bandwidth and storage space is a significant problem in centralized systems when the number of entities increases in magnitude. The proposed system does however provide decentralized access to information, which scales better in comparison. This can be proved quantitatively by looking at the sum of all context information which a centralized system must manage in its storage. Equation 3 shows the total amount of stored context C_{stored} in the central database. This can be compared to equation 4, which shows the amount stored on each entity in the proposed system. From this it is possible to deduce that the amount of data in a central database will become unmanageable when the amount of total entities increases in magnitude. The proposed system will however still have a manageable set of stored data, because the stored amount is only based on the relevant number of entities, not the total number of entities in the system. To be concrete, if a system has in total 100 000 entities, with ten context values each, but only fifty of these entities can be considered relevant for a particular application. The total stored size for a centralized system would then be 1 000 000 entries, compared to 510 entries in the proposed system.

$$C_{stored} = \sum_{n=1}^{total\ entities} n * C_{remote} \tag{3}$$

$$C_{stored} = C_{local} + \sum_{n=1}^{relevant\ entities} n * C_{remote} \tag{4}$$

The same problem occurs when comparing the delays for an application querying the stored context information. The delay $D_{application}$ for centralized access can be seen in equation 5, where the transmission delay over the Internet $d_{transmission}$ is added twice on top of the database query time $d_{central\ database}$. In the proposed architecture the same application access is performed locally, see equation 6. Thus the delay is isolated to the database query inside the local database $d_{local\ database}$, which was previously proven to also contain a much smaller dataset. Hence, the centralized lookup will always be two transmission delays longer than the delay for the local database, regardless of centralized location and replication.

$$D_{application} = d_{transmission} + d_{central\ database} + d_{transmission} \tag{5}$$

$$D_{application} = d_{local\ database} \tag{6}$$

The proposed system also has a lower propagation delay $D_{propagation}$ for when context information changes. This can be seen in equation 7 and equation 8, which shows the delay from the point when the context is updated to that when it has arrived at the application. In centralized systems it is apparent that the context information must be routed through the centralized point, which induces an additional transmission delay over the Internet in comparison to peer-to-peer dissemination. To be concrete, the propagation delay of a centralized system will always add one additional transmission delay over the Internet, because it has to relay the information.

$$D_{propagation} = d_{transmission} + d_{central\ database} + d_{transmission} \tag{7}$$

$$D_{propagation} = d_{transmission} + d_{local\ database} \tag{8}$$

This problem can also be found when studying the required workload by the centralized component compared to the distributed workload of each node in the proposed system. The total workload required to create one view of a schema for each end application in a centralized system can be seen in equation 9, this can be compared to equation 10 which shows the required workload on each end node in the proposed system. However, one important thing to note is that the total workload will be the same in both systems i.e. the workload of the centralized system will be equal to the sum of workload by all nodes in the proposed system.

$$Workload_{central} = \sum_{n=0}^{total\ entities} n * (\sum_{m=0}^{related\ entities} m * C_{remote}) \tag{9}$$

$$Workload_{per\ node} = \sum_{n=0}^{related\ entities} n * C_{remote} \tag{10}$$

4.2 Comparison to Semi Distributed Architectures

In comparison, both the proposed system and semi distributed systems have the propagation delay shown in equation 8. However, since a federated broker still becomes a centralized component, it will scale poorly when performing queries on the whole dataset. Thus, semi distributed systems scales as a centralized system, as in equation 5 for such actions. Furthermore, because the centralized component will require complete knowledge of the system to provide this service, the system will have to store both centrally in the central component according to equation 3 and remotely on each entity as in equation 4. Thus, the system has a very large total amount of stored context information.

4.3 Comparison to other Fully Distributed Architectures

The proposed system is fully distributed, but it has advantages over alternative fully distributed systems. Related systems such as SOFIA and COSMOS build on much more cumbersome protocols, which have a larger overhead than the DCXP protocol. The other distributed systems also offer direct dissemination of context between entities in a similar manner, but only if the destination is known beforehand. Thus, such systems would have to communicate with all the entities on the system in order to create a relevance view. This can be defined as the total signaling required to create a view and is denoted in equation 11. This can be compared to the proposed architecture which would only requires a smaller amount as in equation 12, because it can limit the amount of entities based on context proximity. To be concrete, given the same system as before with 100 000 entities having ten context values each and fifty relevant entities for a particular application. The total signaling in related distributed systems would be 200 000 transmissions and 100 000 local database lookups, where in the proposed system this is limited to 100 transmissions and 50 database lookups. However, it is important to note that the imposed delay of the communication do not cumulatively sum to the total delay it takes for creating a view, since the communication can be performed concurrently among all entities.

$$S_{total} = \sum_{n=0}^{total\ entities} n * (S_{transmission} + S_{local\ database} + S_{transmission}) \tag{11}$$

$$S_{total} = \sum_{n=0}^{related\ entities} n * (S_{transmission} + S_{local\ database} + S_{transmission}) \tag{12}$$

5 Conclusions

This paper proposed an approach to create evolving context information derived from both global and local sensor information. For this we created a system which is capable of enabling continuously evolving context from sensor based sources, as adaptive views for applications within predictable time bounds. Our system utilizes per entity unique context objects called context schemas, which are evolved by data-mining agents. These agents provide continuous evolution by concurrently acquiring context from local and remote sources, while traversing context relations to find new and relevant sources of context. The architecture can thus address the requirements defined in section 1 for the demanded system. Requirement 1 concerned ubiquitous access to context, is fulfilled because the proposed system can provide ubiquitous and global context information. It also provides it as current, relevant, and continuously changing views from context schemas, which was demanded by requirement 5. Requirement 2 concerned the support for large amount of entities and requirement 3 concerned dissemination within predictable time bounds, which are both fulfilled by utilizing the well scaling DCXP network and its context user-agents. In detail, the support for a large number of mobile entities comes from the logarithmic scaling and the context exchange within predicable time bounds is provided by the peer-to-peer dissemination. Furthermore, the system also fulfills requirement 4 which concerned open ended searches, because it can perform these searches on the context dataset by traversing the available relations.

The proposed architecture is implemented as a proof of concept prototype based on a scenario with traveling cars exchanging road characteristics, utilizing location based sensors which is continuously updating. It is currently being planned for field trials, which will evaluate and measure the required properties. Such as initial seeding, propagation delay, workload, application delay, robustness, battery consumption, and scalability. The architecture will thus become intensively evaluated, to prove that it actually can manage real sensors and mobile entities with volatile connections. However, the need for context-aware systems that can provide continuously evolving context is going to be required for future context-aware applications, in particularly for the Internet of Things.

References

1. Hong, J., Suh, E., Kim, S.J.: Context-aware systems: A literature review and classification. Expert Systems with Applications 36(4), 8509–8522 (2009)
2. 3GPP, TS 24.141: Presence service using the IP Multimedia (IM) Core Network (CN) subsystem; Stage 3. 3GPP (December 2009), `http://www.3gpp.org/ftp/Specs/html-info/24141.html`
3. Kansal, A., Nath, S., Liu, J., Zhao, F.: Senseweb: An infrastructure for shared sensing. IEEE MultiMedia 14(4), 8–13 (2007)
4. Presser, M., Barnaghi, P.M., Eurich, M., Villalonga, C.: The SENSEI project: integrating the physical world with the digital world of the network of the future. IEEE Communications Magazine 47(4), 1–4 (2009)
5. Saint-Andre, P.: Extensible Messaging and Presence Protocol (XMPP): Core. IETF, RFC 3920 (2004), `http://www.ietf.org/rfc/rfc3920.txt`
6. Project Serenoa: Multi-Dimensional Context-Aware Adaptation of Service Front-Ends, "Context Aware Design Space and Context Aware Reference Framework," FP7 ICT 258030, Deliverable 2.1.1 (2011)
7. Klemettinen, M.: Enabling Technologies for Mobile Services: The MobiLife Book. John Wiley and Sons Ltd. (2007)
8. Raz, D., Juhola, A., Serrat-Fernandez, J., Galis, A.: In: Hutchison, D. (ed.) Fast and Efficient Context-Aware Services. Wiley, Chichester (2006)
9. Koumaras, H., Negrou, D., Liberal, F., Arauz, J., Kourtis, A.: ADAMANTIUM project: Enhancing IMS with a PQoS-aware multimedia content management system. In: International Conference on Automation, Quality and Testing, Robotics, vol. 1, pp. 358–363 (2008)
10. Kanter, T., Österberg, P., Walters, J., Kardeby, V., Forsström, S., Pettersson, S.: The mediasense framework. In: Proceedings of 4th IARIA International Conference on Digital Telecommunications (ICDT), Colmar, France, pp. 144–147 (July 2009)
11. Toninelli, A., Pantsar-Syväniemi, S., Bellavista, P., Ovaska, E.: Supporting context awareness in smart environments: a scalable approach to information interoperability. In: Proceedings of the International Workshop on Middleware for Pervasive Mobile and Embedded Computing, pp. 1–4. ACM (2009)
12. Bellavista, P., Montanari, R., Tibaldi, D.: COSMOS: A Context-Centric Access Control Middleware for Mobile Environments. In: Horlait, E., Magedanz, T., Glitho, R.H. (eds.) MATA 2003. LNCS, vol. 2881, pp. 77–88. Springer, Heidelberg (2003)
13. Forsström, S., Kardeby, V., Walters, J., Kanter, T.: Location-Based Ubiquitous Context Exchange in Mobile Environments. In: Pentikousis, K., Agüero, R., García-Arranz, M., Papavassiliou, S. (eds.) MONAMI 2010. LNICST, vol. 68, pp. 177–187. Springer, Heidelberg (2011)
14. Dobslaw, F., Larsson, A., Kanter, T., Walters, J.: An Object-Oriented Model in Support of Context-Aware Mobile Applications. In: Cai, Y., Magedanz, T., Li, M., Xia, J., Giannelli, C. (eds.) Mobilware 2010. LNICST, vol. 48, pp. 205–220. Springer, Heidelberg (2010)
15. Antifakos, S., Schiele, B., Holmquist, L.: Grouping mechanisms for smart objects based on implicit interaction and context proximity. In: Adjunct Proceedings of International Conference on Ubiquitous Computing (Ubicomp), Seattle, USA. Citeseer (2003)
16. Kanter, T., Pettersson, S., Forsstrom, S., Kardeby, V., Norling, R., Walters, J., Osterberg, P.: Distributed context support for ubiquitous mobile awareness services. In: Fourth International Conference on Communications and Networking in China, ChinaCOM 2009, pp. 1–5 (August 2009)